Overview of the Global Arrays Parallel Software Development Toolkit

Bruce Palmer

Jarek Nieplocha, Manoj Kumar Krishnan, Vinod Tipparaju, Harold Trease

Pacific Northwest National Laboratory

Overview

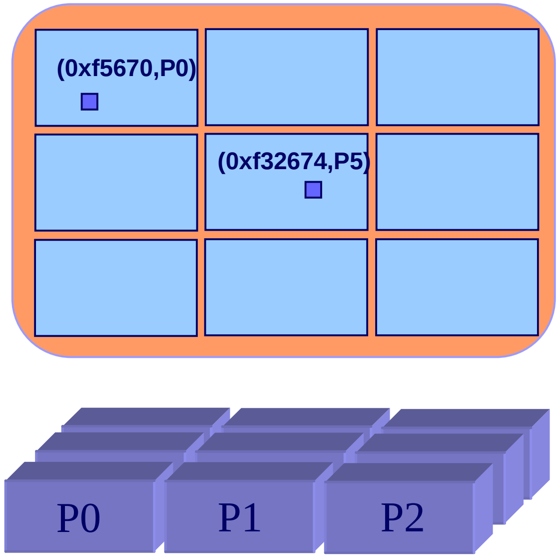
- Background
- Programming Model
- Core Capabilities
- Applications
- Summary

Distributed Data vs Shared Memory

Distributed Data:

Data is explicitly associated with each processor, accessing data requires specifying the location of the data on the processor and the processor itself.

Data locality is explicit but data access is complicated. Distributed computing is typically implemented with message passing (e.g. MPI)

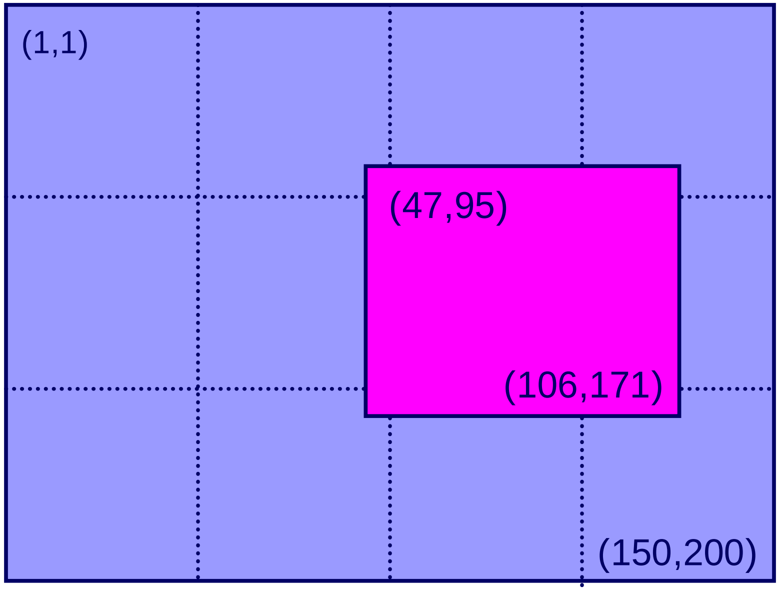


Distributed Data vs Shared Memory (Cont).

Shared Memory:

Data is an a globally accessible address space, any processor can access data by specifying its location using a global index

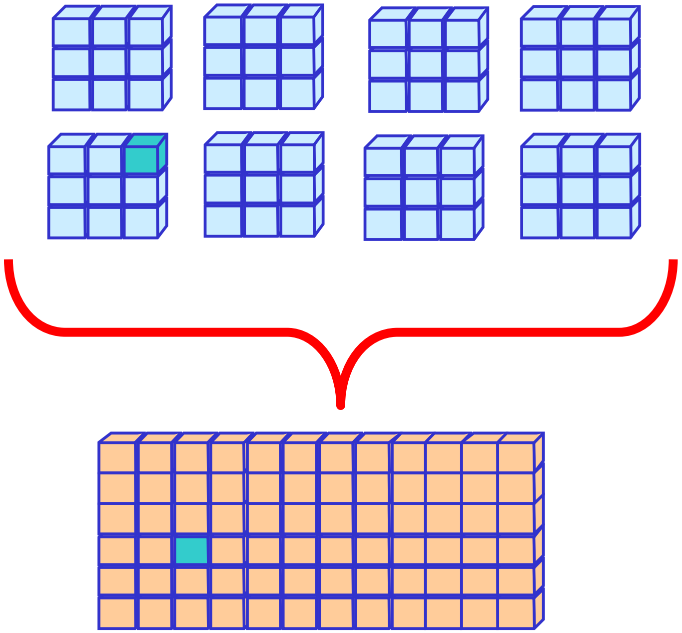
Data is mapped out in a natural manner (usually corresponding to the original problem) and access is easy. Information on data locality is obscured and leads to loss of performance.



Global Arrays

Distributed dense arrays that can be accessed through a shared memory-like style

Physically distributed data



Global Address Space

single, shared data structure/ global indexing

e.g., access A(4,3) rather than buf(7) on task 2

Global Arrays (cont.)

- Shared memory model in context of distributed dense arrays
- Much simpler than message-passing for many applications
- Complete environment for parallel code development
- Compatible with MPI
- Data locality control similar to distributed memory/message passing model
- Extensible
- Scalable

Remote Data Access in GA

Message Passing:

identify size and location of data blocks

loop over processors:
    if (me = P_N) then
        pack data in local message buffer
        send block of data to message buffer on P0
    else if (me = P0) then
        receive block of data from P_N in message buffer
        unpack data from message buffer to local buffer
    endif
end loop

copy local data on P0 to local buffer

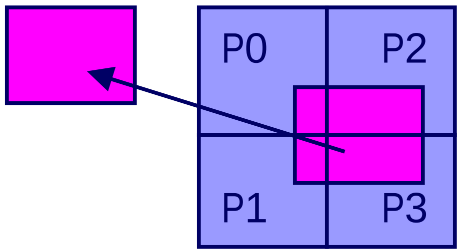
Global Arrays:

NGA_Get(g_a, lo, hi, buffer, ld);

Global Array handle

Global upper and lower indices of data patch

Local buffer and array of strides



Data Locality

What data does a processor own?

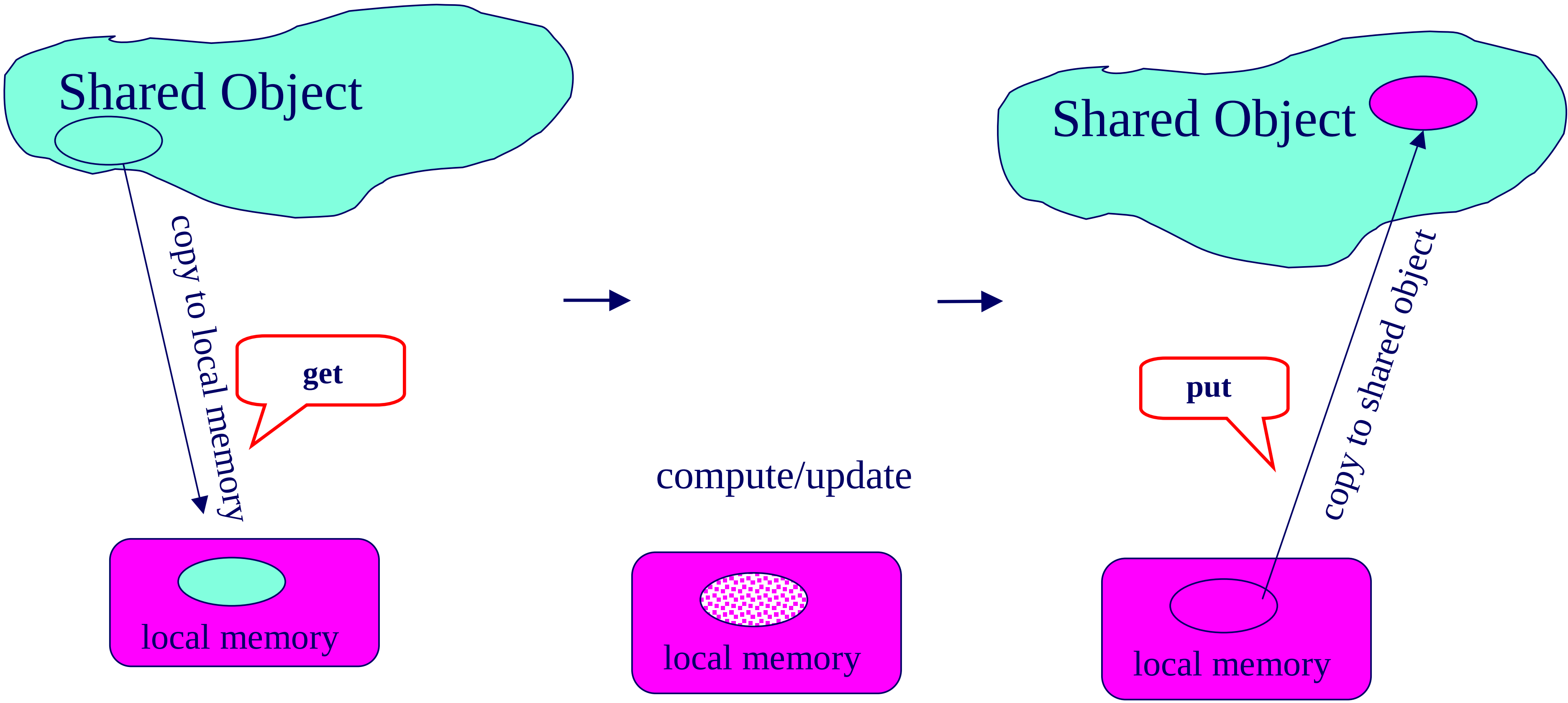
NGA_Distribution(g_a, iproc, lo, hi);

Where is the data?

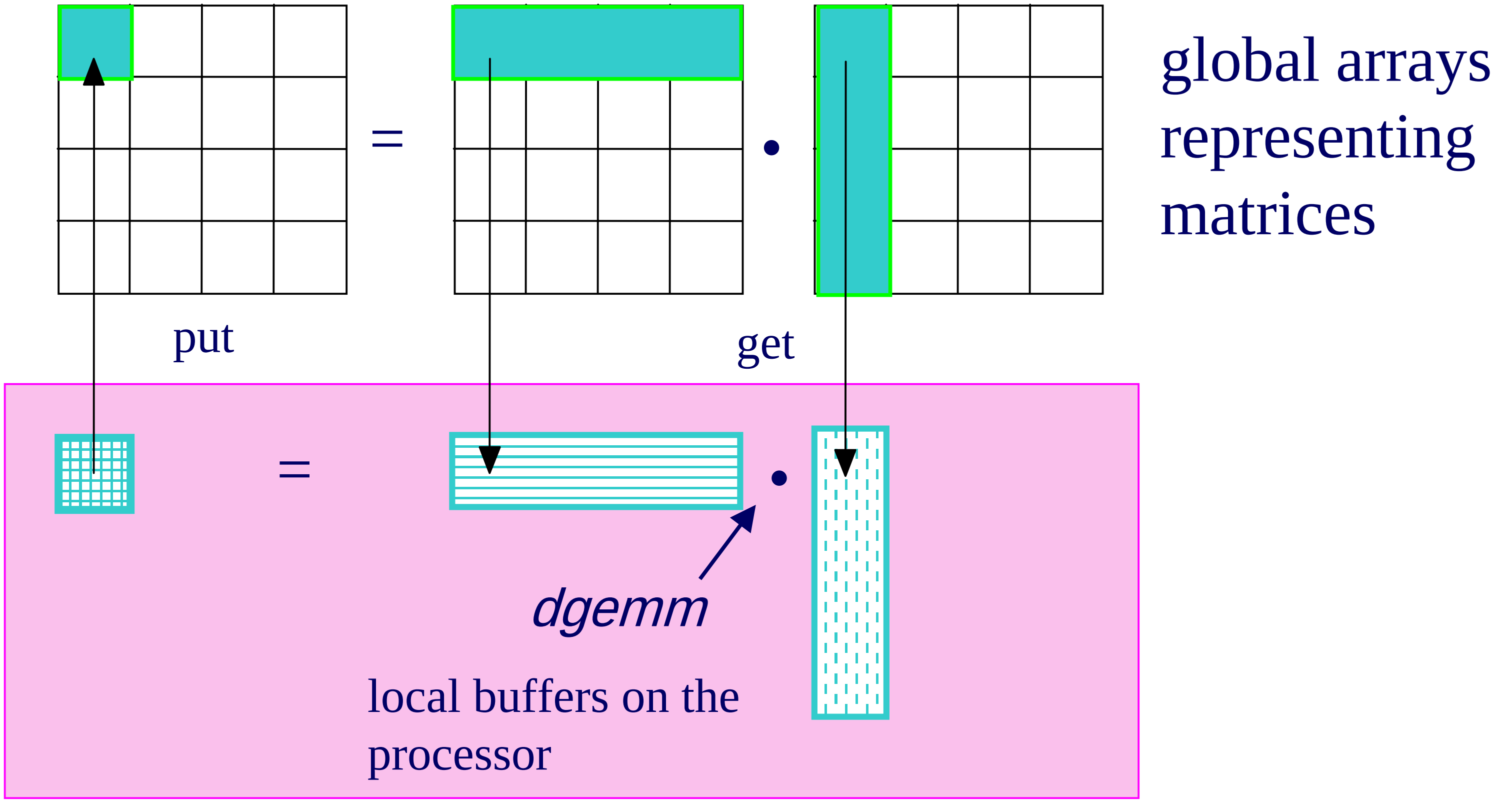
NGA_Access(g_a, lo, hi, ptr, ld)

Use this information to organize calculation so that maximum use is made of locally held data

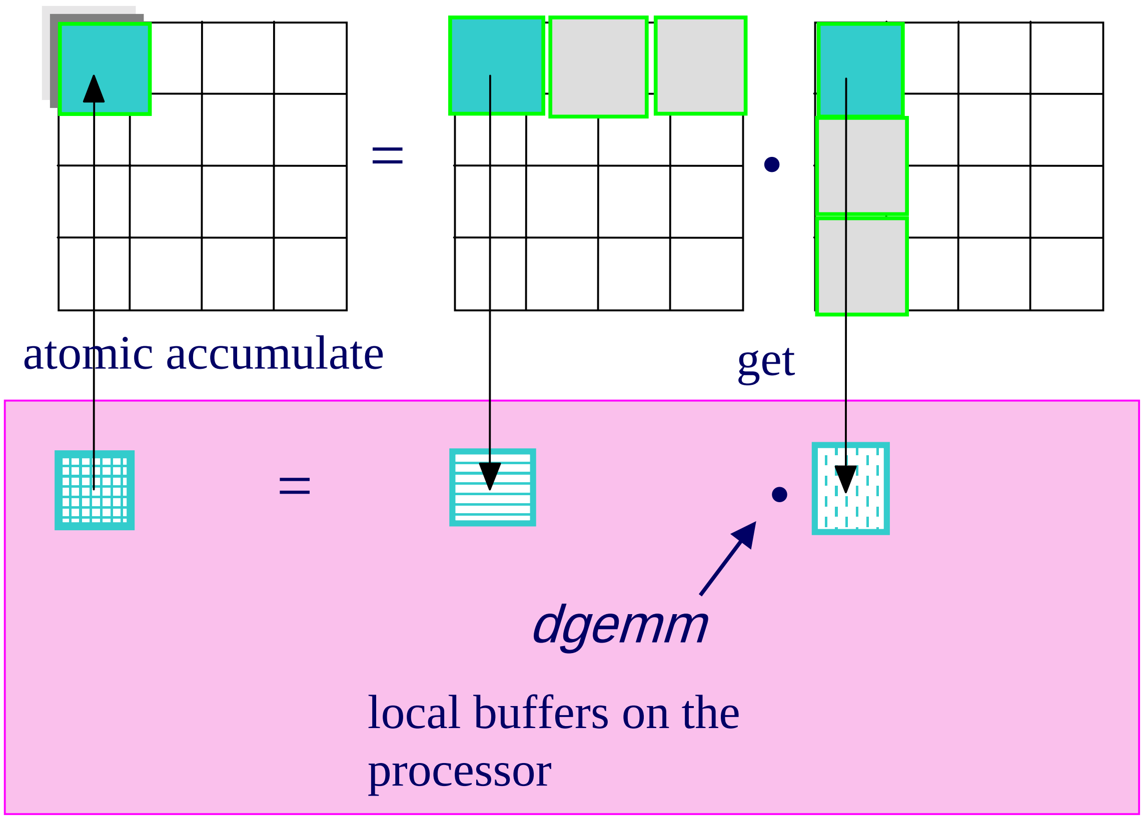
Global Array Model of Computations



Example: Matrix Multiply

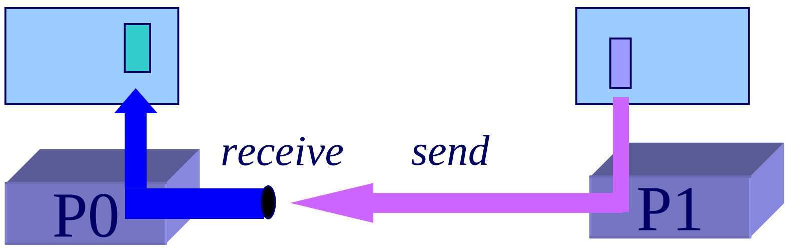


Matrix Multiply
(a better version)

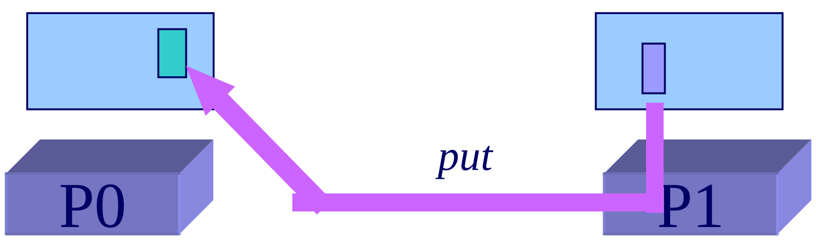


more scalable!
(less memory,
higher parallelism)

One-sided Communication



message passing
MPI



one-sided communication
SHMEM, ARMCI, MPI-2-1S

Message Passing:

Message requires cooperation on both sides. The processor sending the message (P1) and the processor receiving the message (P0) must both participate.

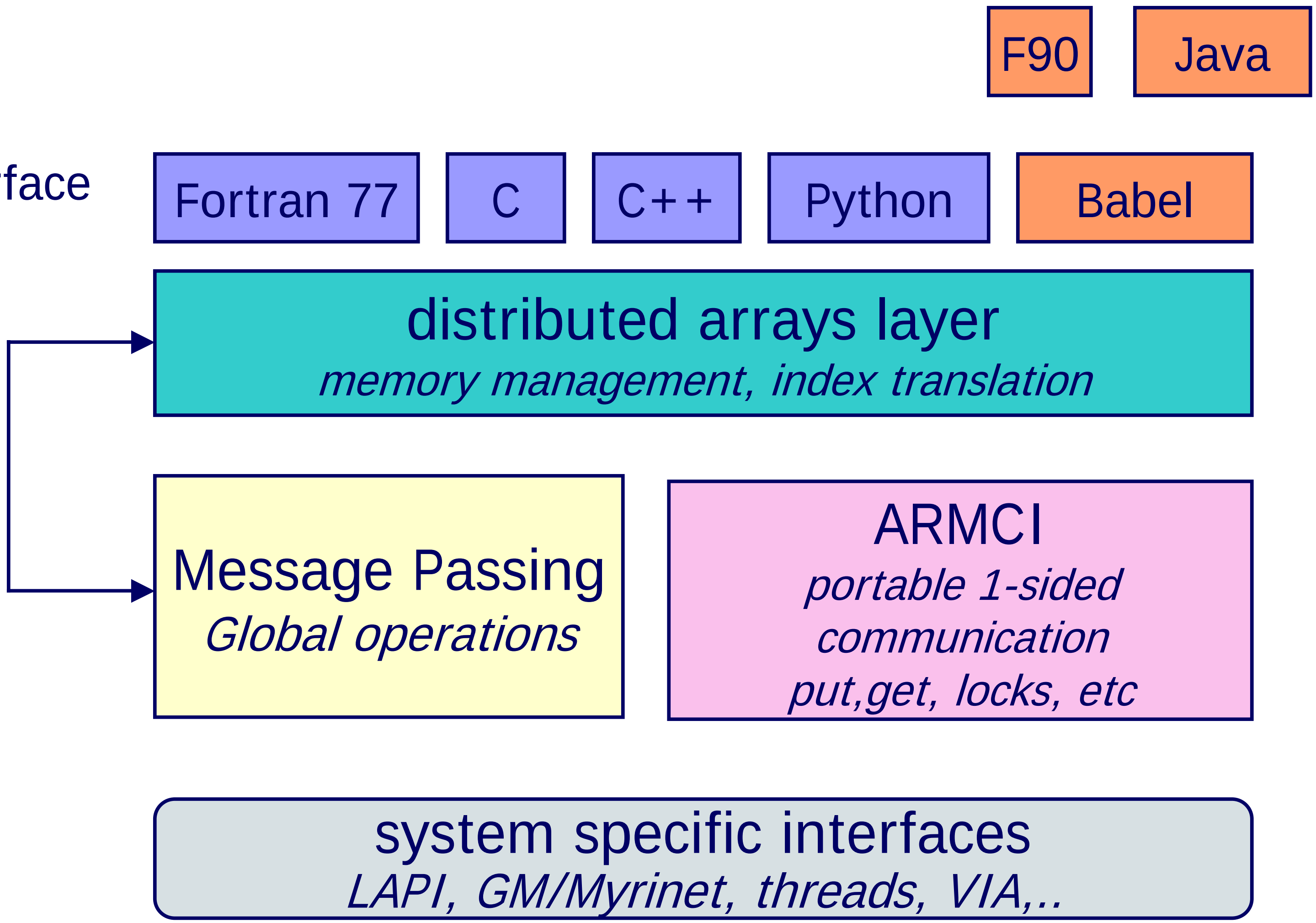
One-sided Communication:

Once message is initiated on sending processor (P1) the sending processor can continue computation. Receiving processor (P0) is not involved.

Structure of GA

Application programming language interface

Global Arrays and MPI are completely interoperable. Code can contain calls to both libraries.



Core Capabilities

- Distributed array library
  - dense arrays 1-7 dimensions
  - four data types: *integer*, *real, double precision*, *double complex*
  - global rather than per-task view of data structures
  - user control over data distribution: regular and irregular
- Collective and shared-memory style operations
  - ga_sync, ga_scale, etc
  - ga_put, ga_get, ga_acc
  - nonblocking ga_put, ga_get, ga_acc
- Interfaces to third party parallel numerical libraries
  - PeIGS, Scalapack, SUMMA, Tao
    - example: to solve a linear system using LU factorization

```
call ga_lu_solve(g_a, g_b)
```

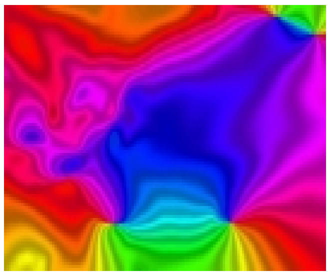
instead of

```
call pdgetrf(n,m, locA, p, q, dA, ind, info)
call pdgetrs(trans, n, mb, locA, p, q, dA,dB,info)
```

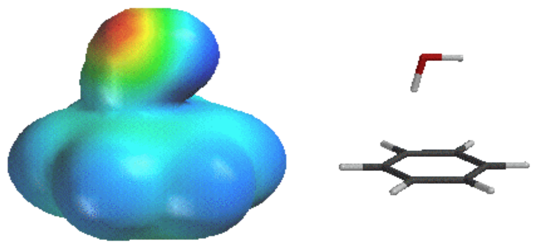
Interoperability and Interfaces

- Language interfaces to Fortran, C, C++, Python
- Interoperability with MPI and MPI libararies
  - e.g., PETSC, CUMULVS
- Explicit interfaces to other systems that expand functionality of GA
  - ScaLAPACK-scalable linear algebra software
  - Peigs-parallel eigensolvers
  - TAO-advanced optimization package

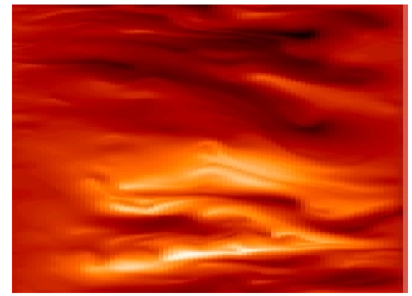
Application Areas



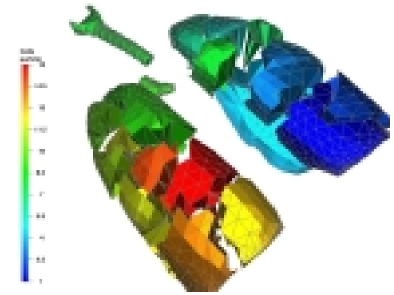
Visualization and image analysis



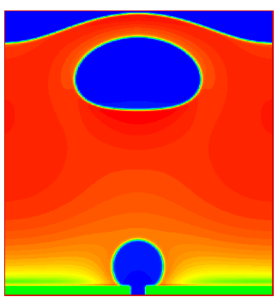
electronic structure



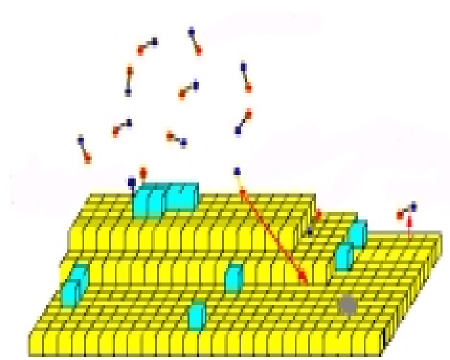
glass flow simulation



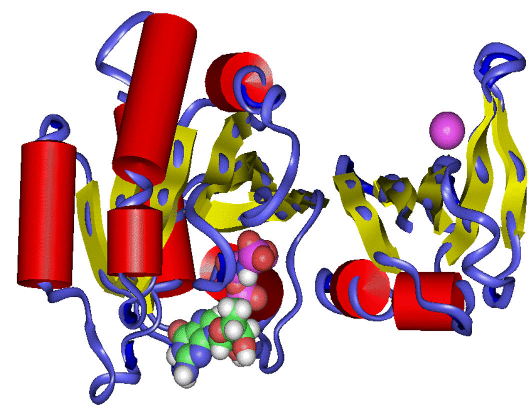
biology



thermal flow simulation



material sciences

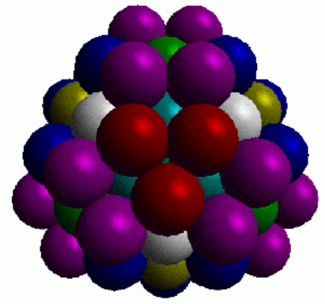


molecular dynamics

Others: financial security forecasting, astrophysics, geosciences

Lennard-Jones Simulation (MD)

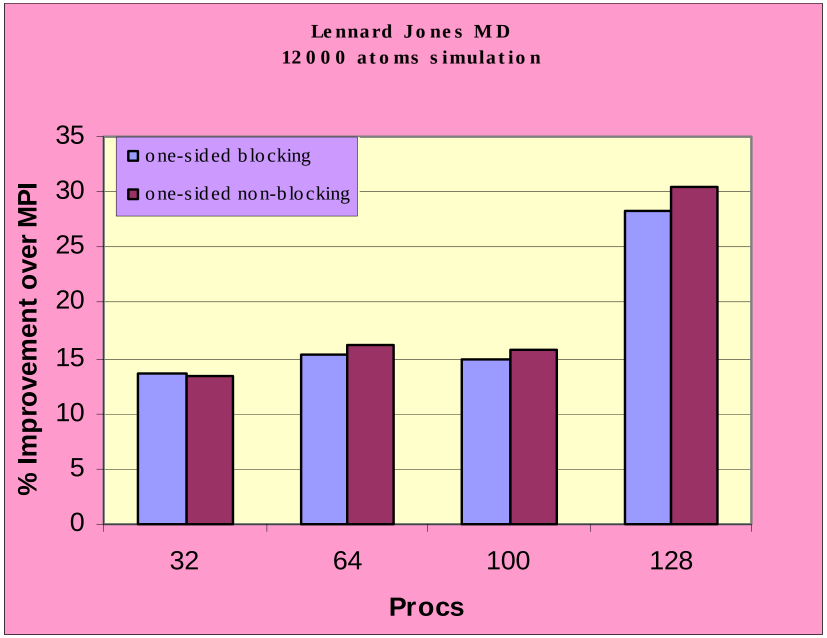
- Molecular Dynamics (MD) Simulation:
  - Simulates particle systems
    - Solids, liquids, gases
    - Biomolecules on Earth
    - Motion of stars, etc.
- GA Implementation:
  - Based on force decomposition
  - Dynamic Load Balancing

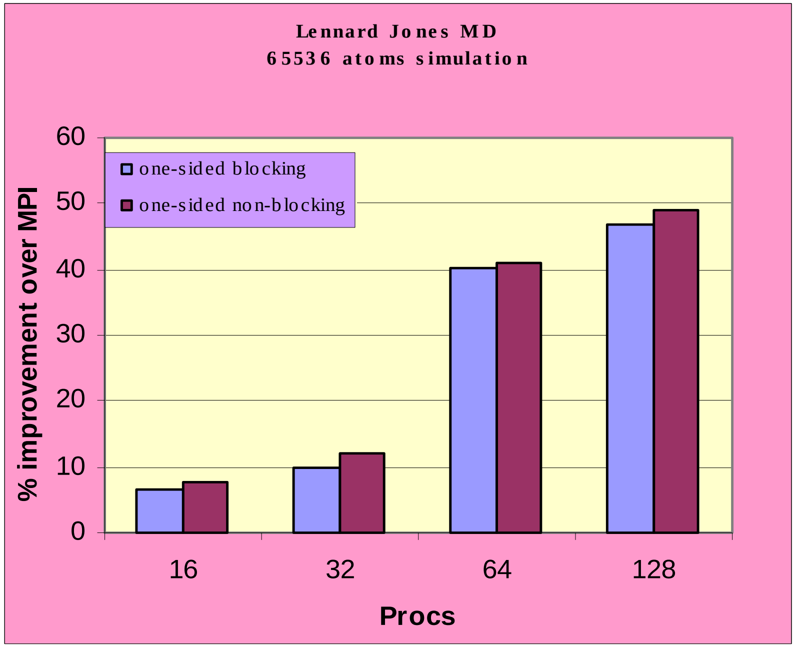


Lennard Jones Potential

$$U(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

MD Performance Results

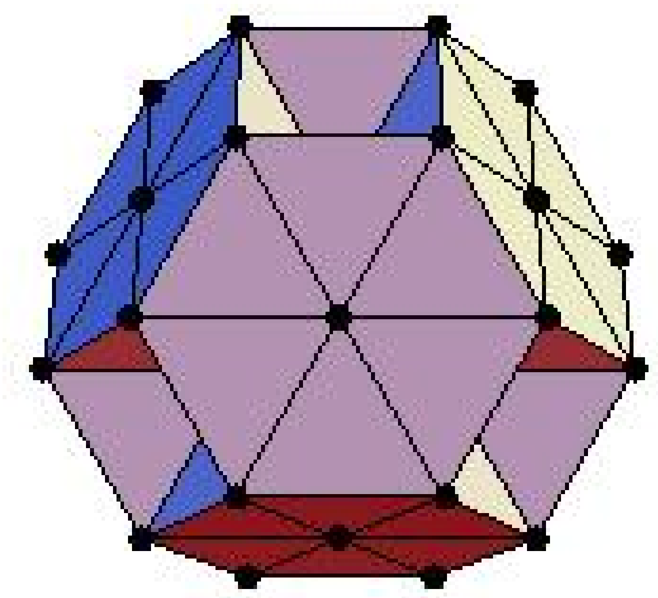




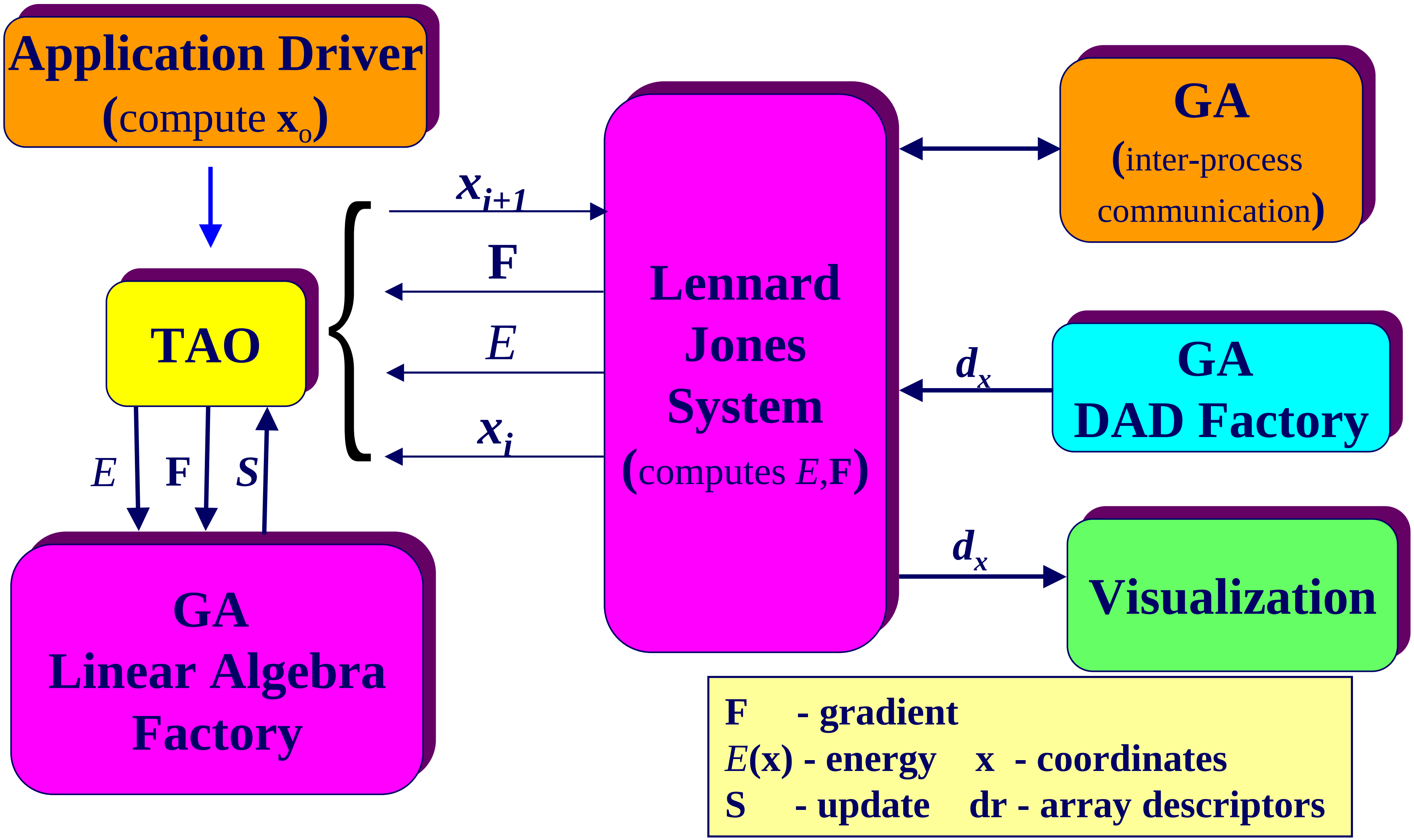
Performance improvement over MPI in molecular dynamics simulation involving 12000 (left) and 65536 (right) atoms

Energy Optimization GA/TAO

- LJMD (Lennard Jones Molecular Dynamics) code
  - computes the function and gradients of the Lennard-Jones clusters (Molecular Conformation) problem
  - Uses GA for communication
  - Uses TAO optimization solvers
- GA/TAO Interoperability
  - GA provides TAO the core linear algebra support for manipulating vectors, matrices, and linear solvers
- GA Distributed Array Descriptor Factory (GA-DADF) provides array descriptors to Visualization component for visualizing molecules



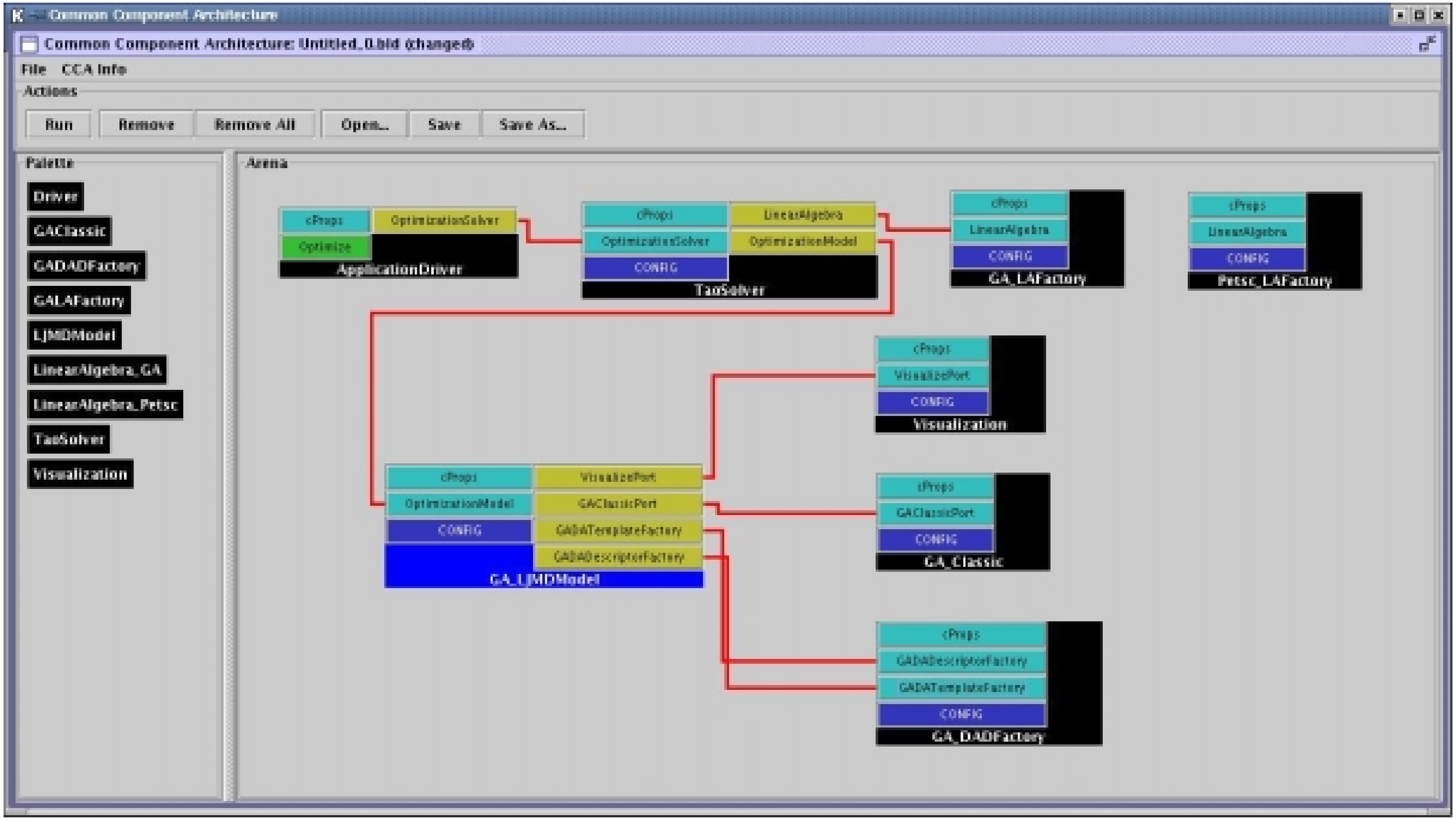
GA/TAO Interaction



Common Component Architecture (CCA)

- Molecular Dynamics of a Lennard-Jones System
- Components Used
  - **LJMDModel** - **Lennard Jones Model**
  - **GA_Classic** - **Native GA Component**
  - **GA_DADFactory** - **GA's Distributed Array Factory**
  - **GA_LAFactory** - **Linear Algebra based on GA**
  - **Petsc_LinearAlgebra** - **Linear Algebra based on Petsc**
  - **TaoSolver** - **Optimization Component**
  - **Visualization** - **Viz Component (OpenGL)**
  - **Driver** - **Driver Component**

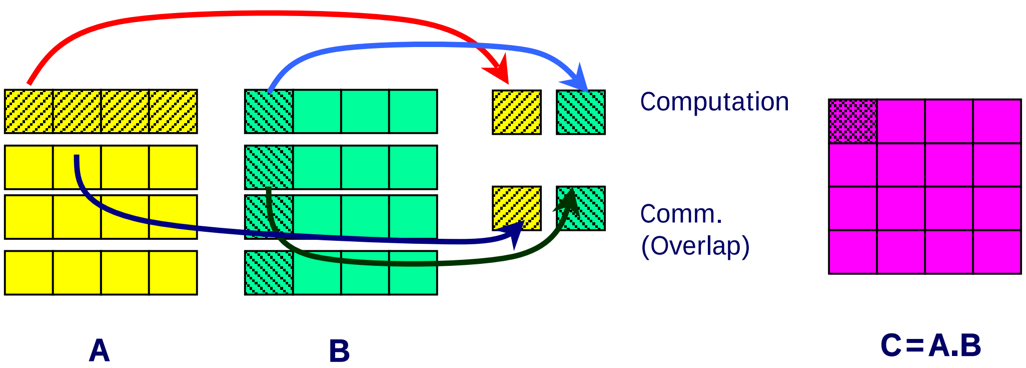
CCA Wiring Diagram



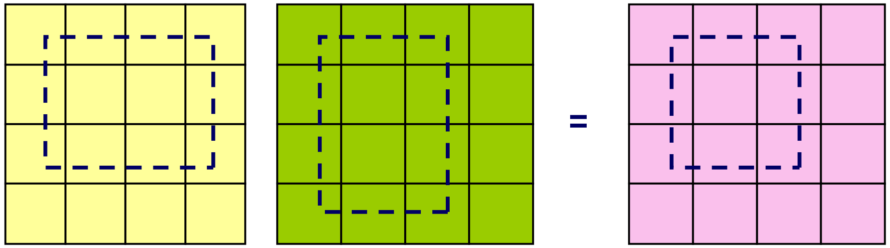
Non-Blocking Communication

- New functionality in GA version 3.3
- Allows overlapping of data transfers and computations
  - Technique for latency hiding
- Nonblocking operations initiate a communication call and then return control to the application immediately
- operation completed locally by making a call to the *wait* routine

SUMMA Matrix Multiplication



Issue NB Get A and B blocks
do (until last chunk)
   issue NB Get to the next blocks
   wait for previous issued call
   compute A*B (sequential dgemm)
   NB atomic accumulate into “C” matrix
done

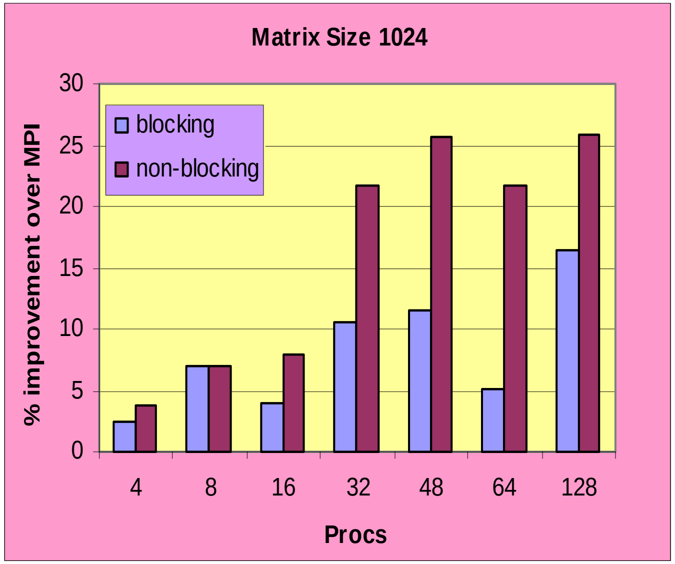


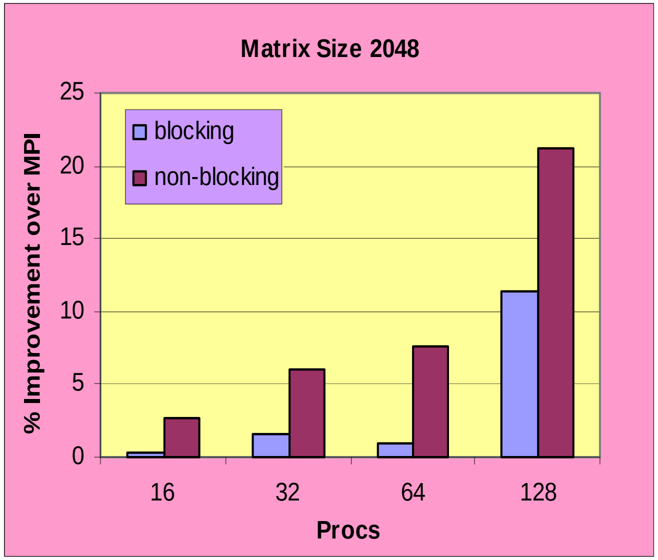
patch matrix multiplication

Advantages:
- Minimum memory
- Highly parallel
- Overlaps computation and communication
  - latency hiding
- exploits data locality
- patch matrix multiplication (easy to use)
- dynamic load balancing

SUMMA Matrix Multiplication: Improvement over MPI

Non-Blocking Communication Performance

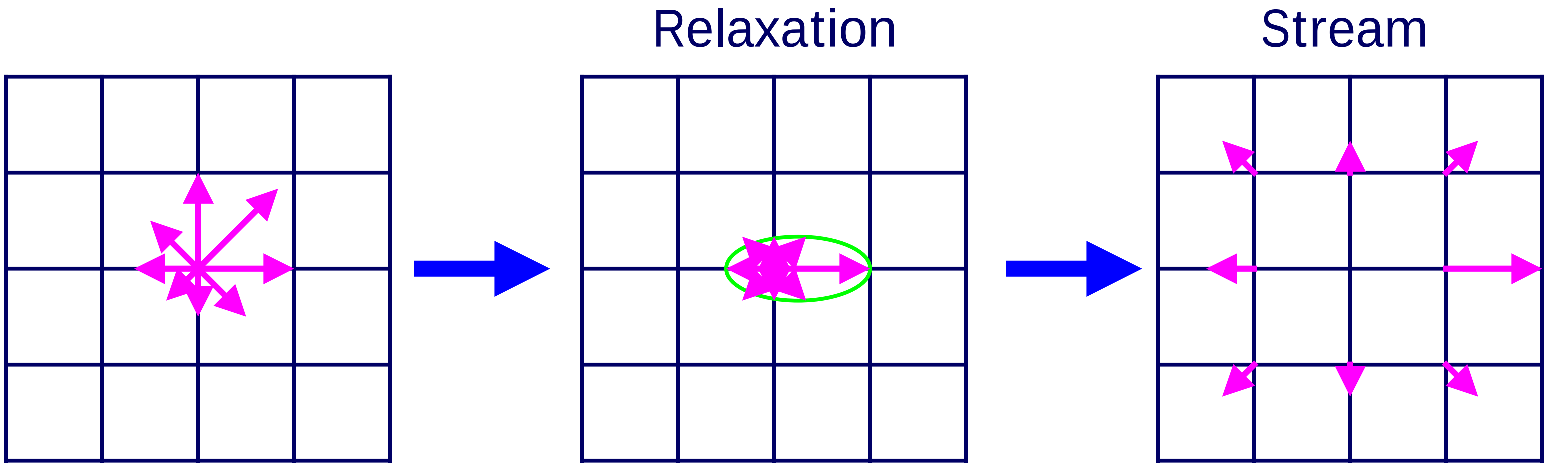




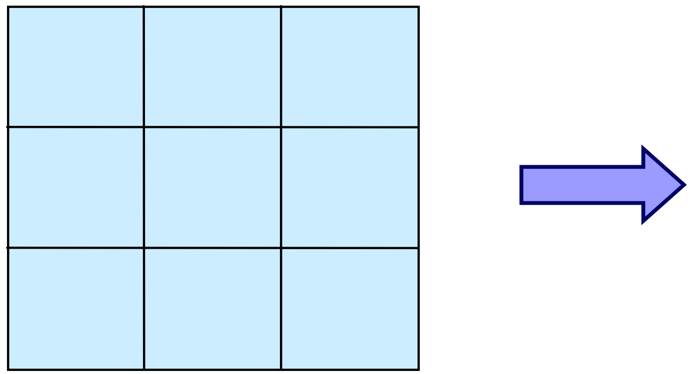
***2.4Ghz P4 Linux cluster, Myrinet-GM interconnect (at SUNY, Buffalo)**

Lattice Boltzmann Simulation

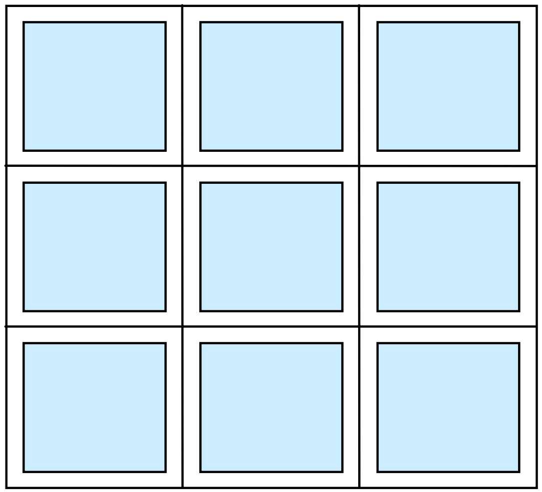
$$f_i(\mathbf{r}+\mathbf{e}_i, t+\Delta t) = f_i(\mathbf{r},t) - \frac{1}{\tau}\left(f_i(\mathbf{r},t) - f_i^{eq}(\mathbf{r},t)\right)$$



Ghost Cells



normal global array



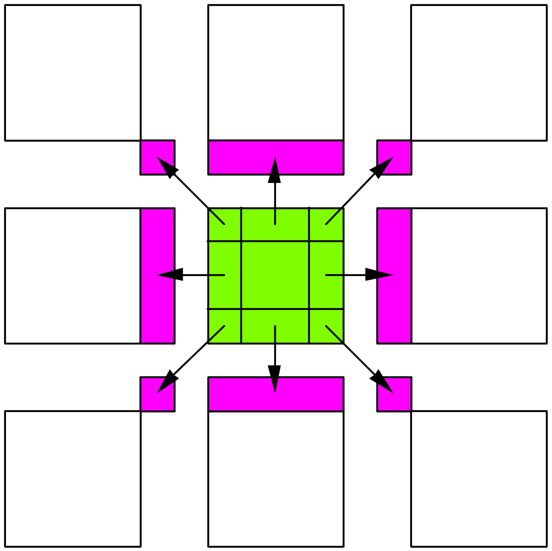
global array with ghost cells

Operations:

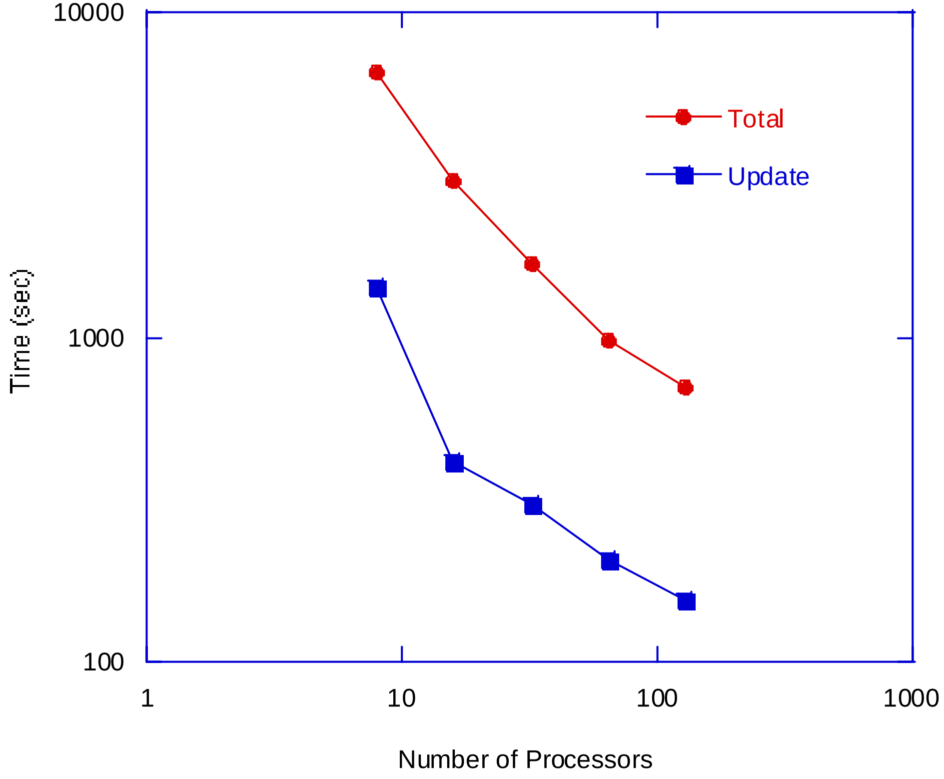
NGA_Create_ghosts	- creates array with ghosts cells
GA_Update_ghosts	- updates with data from adjacent processors
NGA_Access_ghosts	- provides access to “local” ghost cell elements
NGA_Nbget_ghost_dir	- nonblocking call to update ghosts cells

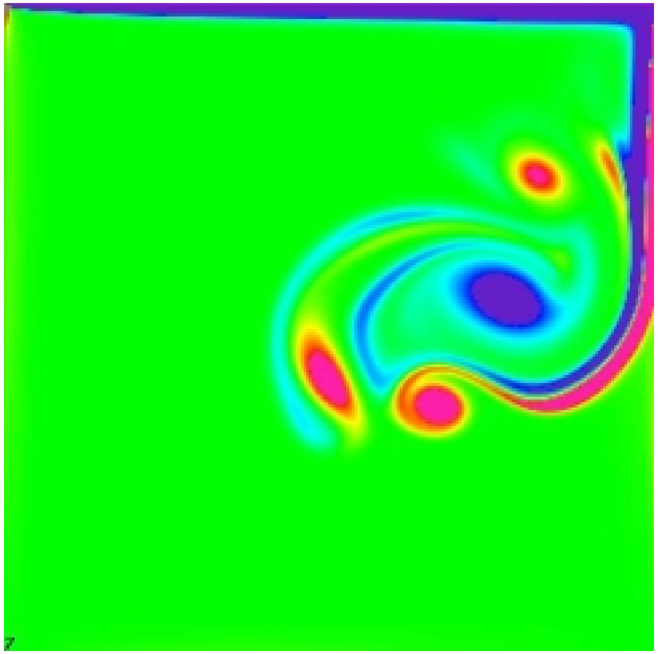
Ghost Cell Update

Automatically update ghost cells with appropriate data from neighboring processors. A multiprotocol implementation has been used to optimize the update operation to match platform characteristics.



Ghost Cell Application Performance

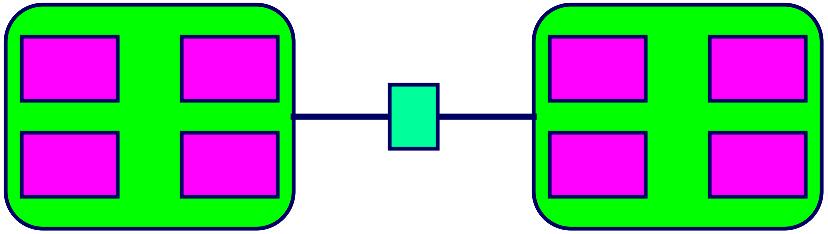


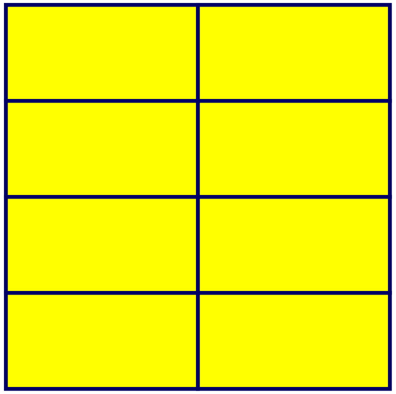


Mirrored Arrays

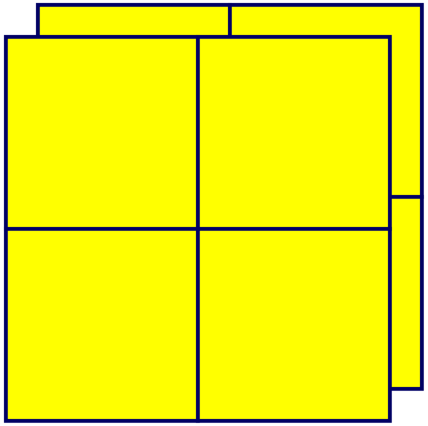
- Create Global Arrays that are replicated between SMP nodes but distributed within SMP nodes
- Aimed at fast nodes connected by relatively slow networks (e.g. Beowulf clusters)
- Use memory to hide latency
- Most of the operations supported on ordinary Global Arrays are also supported for mirrored arrays
- Global Array toolkit augmented by a merge operation that adds all copies of mirrored arrays together
- Easy conversion between mirrored and distributed arrays

Mirrored Arrays (cont.)

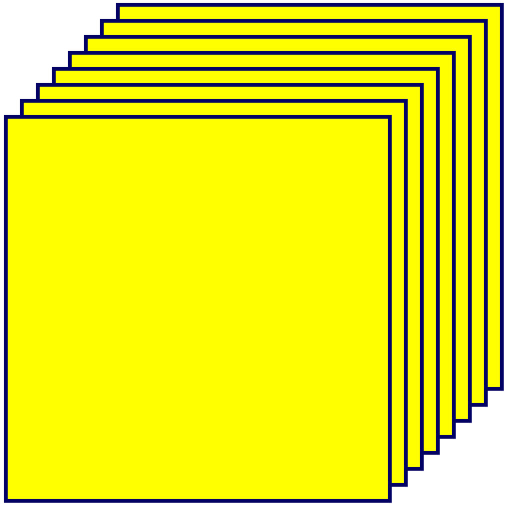




Distributed

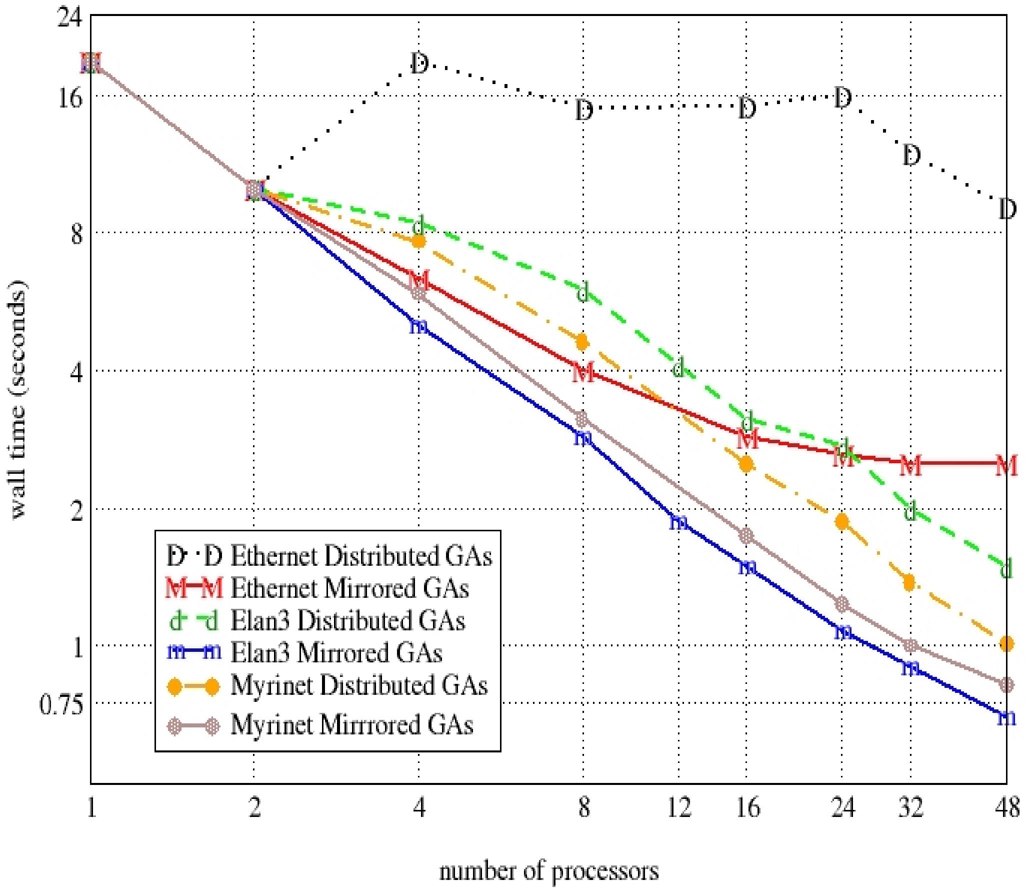


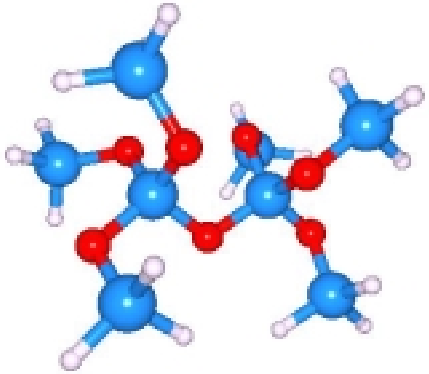
Mirrored



Replicated

NWChem DFT Calculation

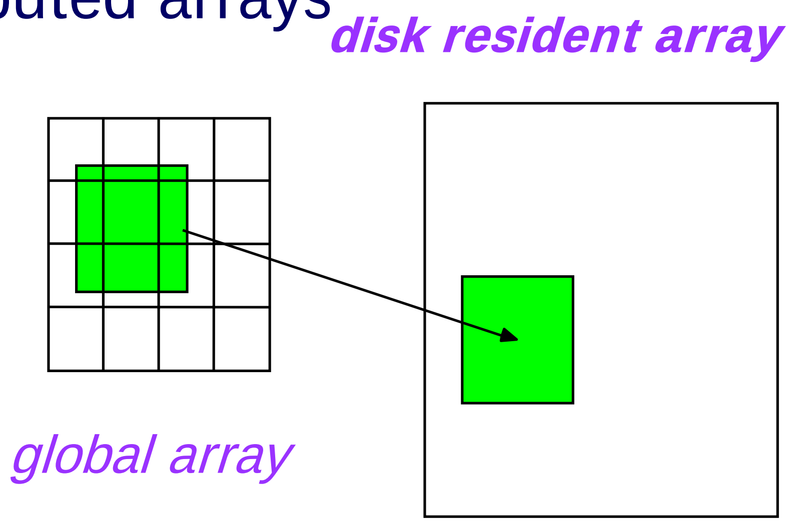




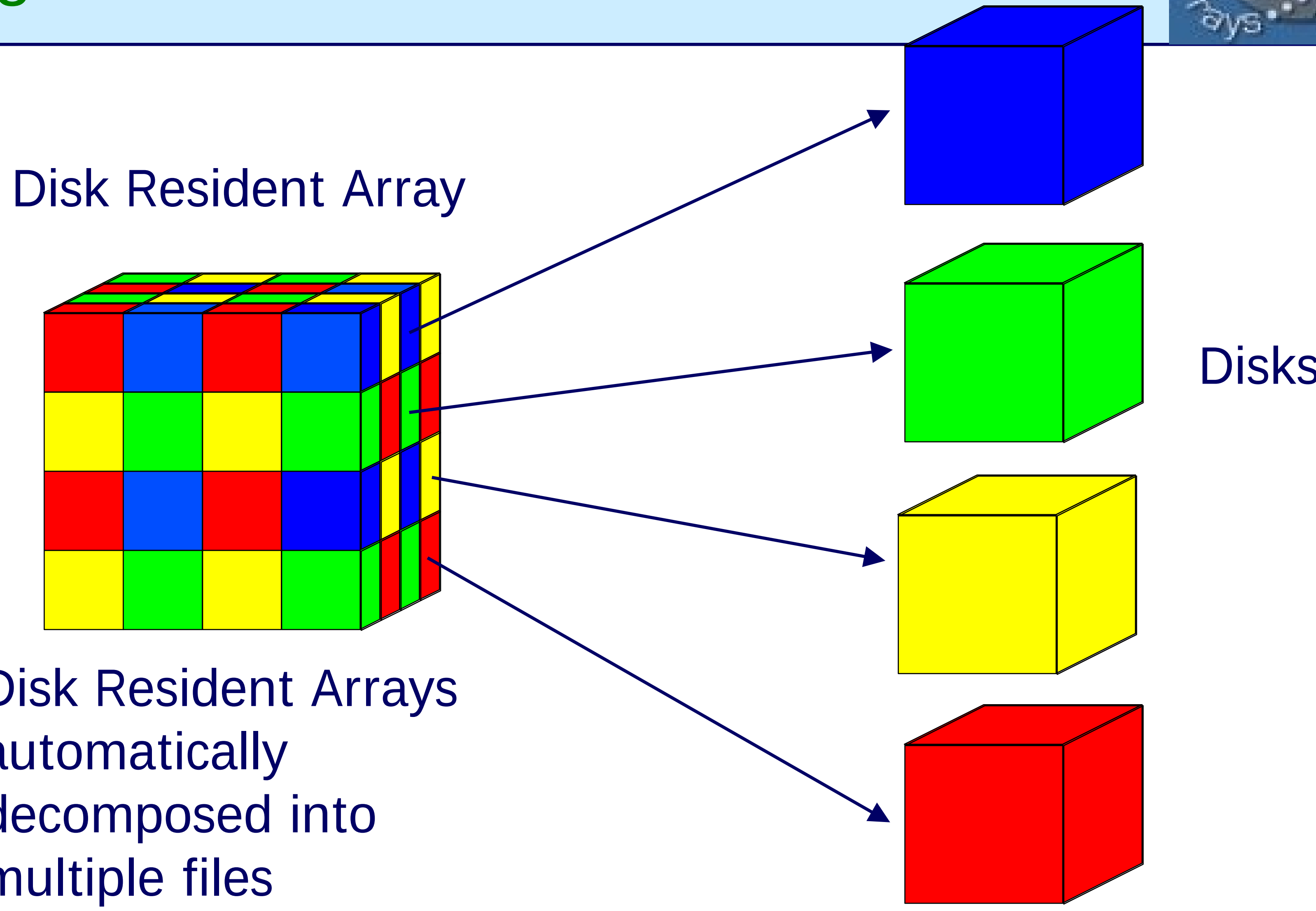
http://www.emsl.pnl.gov/docs/nwchem

Disk Resident Arrays

- Extend GA model to disk
  - system similar to Panda (U. Illinois) but higher level APIs
- Provide easy transfer of data between N-dim arrays stored on disk and distributed arrays stored in memory
- Use when
  - Arrays too big to store in core
  - checkpoint/restart
  - out-of-core solvers



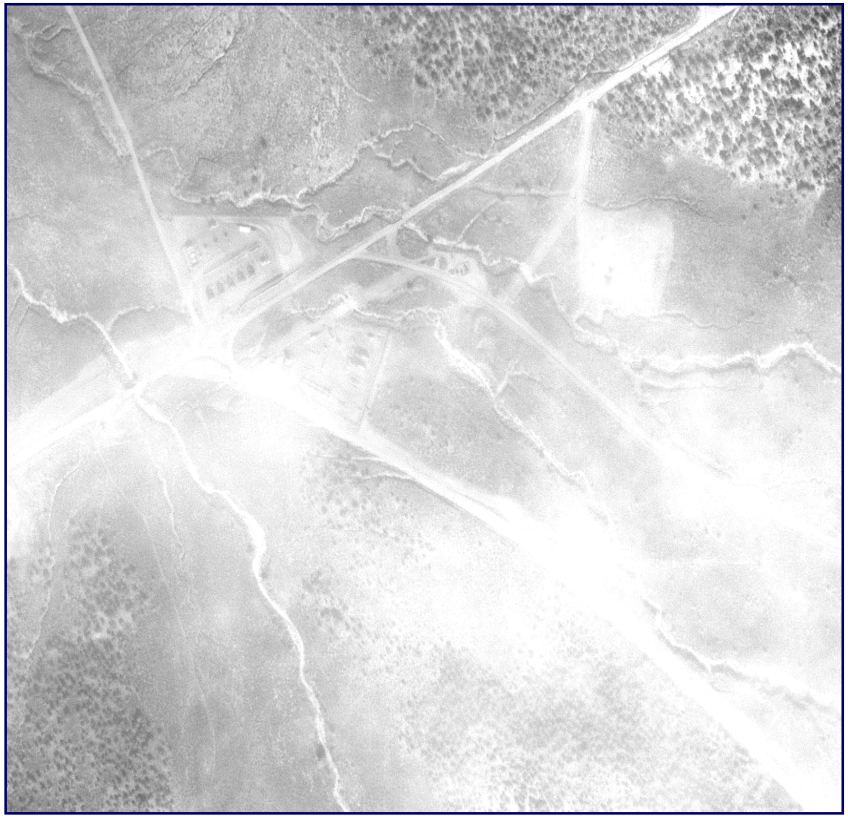
High Bandwidth Read/Write



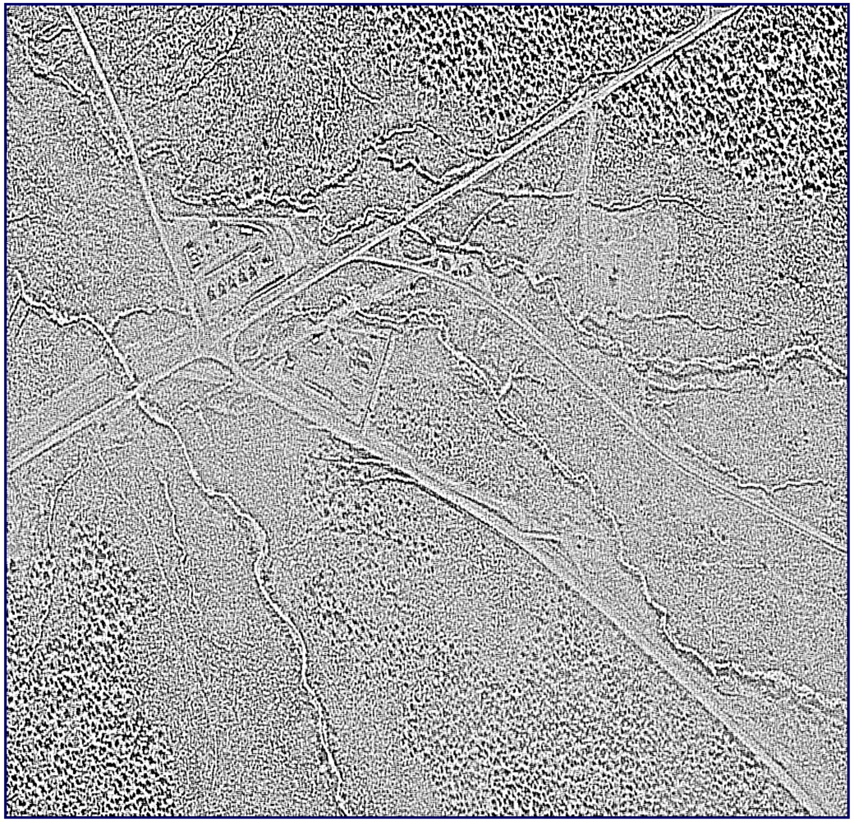
Disk Resident Arrays automatically decomposed into multiple files

Image Processing Application
PiCEIS

Parallel Computational Environment for Imaging Science



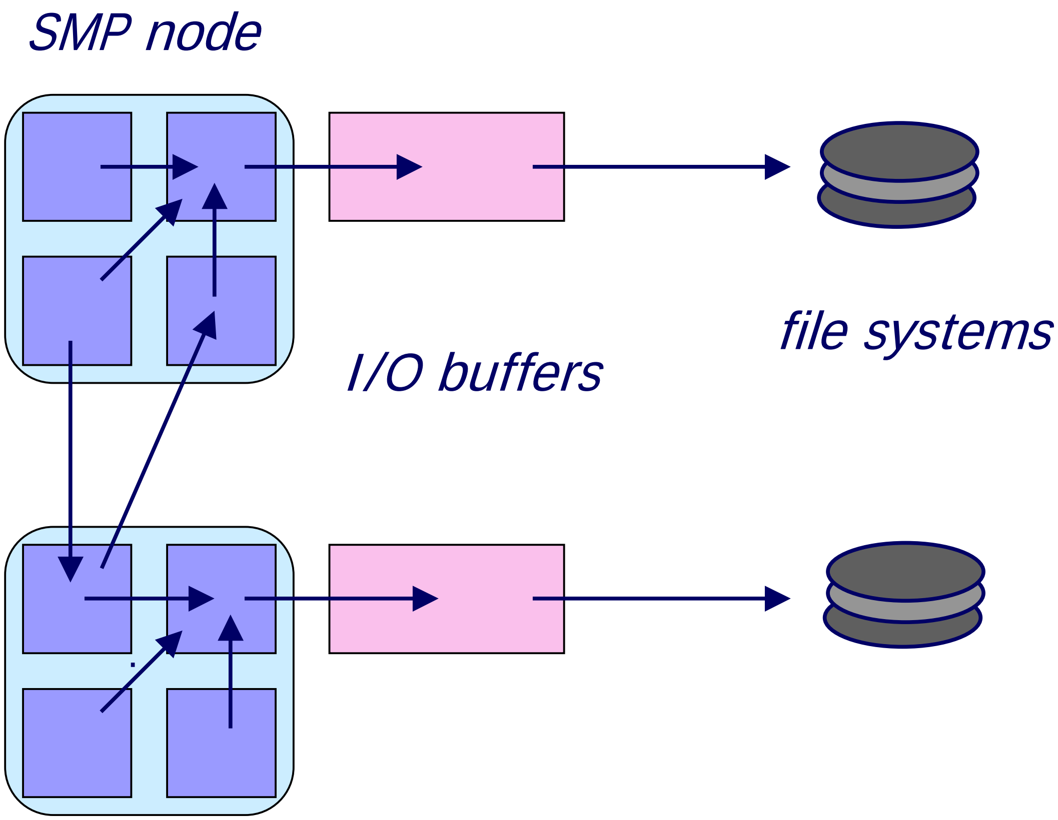
IKONOS

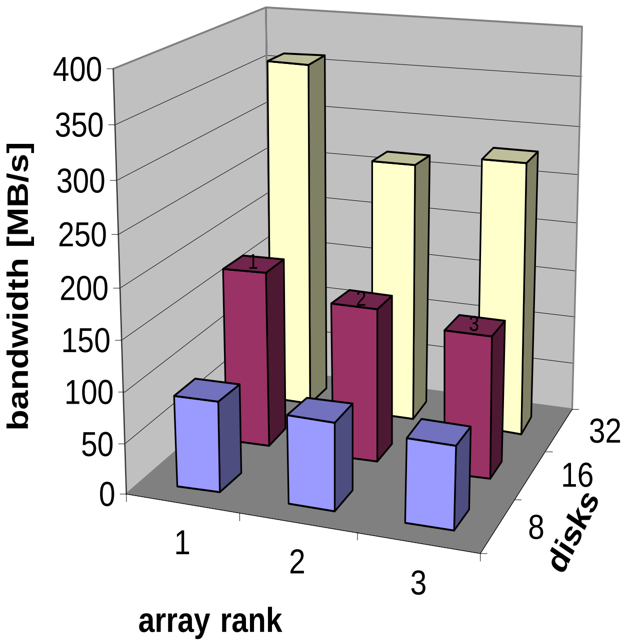


ISAT Texture (George He)

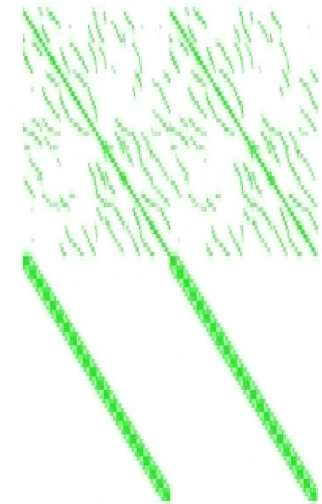
erjurrus@pnl.gov

Scalable Performance of DRA



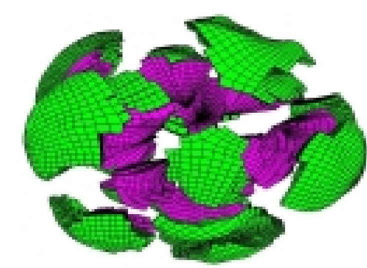


Sparse data managment



- Sparse arrays can be implemented with
  - 1-dimensional global arrays
    - Nonzero elements, row and/or index arrays
  - Set of new operations follow Thinking Machines CMSSL
    - Enumerate
    - Pack/unpack
    - Binning (NxM mapping)
    - 2-key binning/sorting functions
    - Scatter_with_OP, where OP={+,min,max}
    - Segmented_scan_with_OP, where OP={+,min,max,copy}
- Adopted in NWPhys/NWGrid AMR package

  http://www.emsl.pnl.gov/nwgrid



Related Programming Tools

- Co-Array Fortran
  - Distributed Arrays
  - One-Sided Communication
  - No Global View of Data
- UPC
  - Model Similar to GA but only applicable to C programs
  - Global Shared Pointers could be used to implement GA functionality
    - C does not really support multi-dimensional arrays
- High level functionality in GA is missing from these systems

Summary

- The idea has proven very successful
  - efficient on a wide range of architectures
    - core operations tuned for high performance
  - library substantially extended but all original (1994) APIs preserved
  - increasing number of application areas
- Supported and portable tool that works in real applications
- Future work
  - Fault tolerance

Major Milestones

- 1994 - 1st public release of GA
- 1995 - Metacomputing (grid) extensions of GA
- 1996 - DRA, parallel I/O for GA programs developed
- 1997 - development of ARMCI started
- 1998 - GA rewritten to use ARMCI
- 1999 - GA 3.0 released, n-dimensional arrays
- 2000 - periodic one-sided operations
- 2001 - support for sparse data management
- 2002 - ghost cell operations, n-dim DRA
- 2003 – mirrored arrays, improved matrix multiply, non-blocking get operations

Source Code and More Information

- Version 3.3 available in beta release
- Homepage at http://www.emsl.pnl.gov:2080/docs/global/
- Platforms (32 and 64 bit)
  - IBM SP
  - Cray T3E, SV1, X1
  - Linux Cluster with Ethernet, VIA, Myrinet, Infiniband, or Quadrics
  - Solaris
  - Fujitsu
  - Hitachi
  - NEC
  - HP
  - Windows